# Assignment 6

### ****Interoperability trilemma****

*We talked before about scalability trilemma where L1 must sacrifice either decentralization, security or scalability. In a cross chain world there is also a famous interoperability trilemma.*

*Interoperability protocols can only have two of the following three properties:*

* ***Trustlessness***
* ***Extensibility***
* ***Generalizability***

1. *Explain each aspect of the interoperability trilemma. Provide an example of a bridge protocol explaining which trade-offs on the trilemma the bridge makes.*

**[ANSWER]**

图表

描述已自动生成

* **Trustlessness: maximizing the security of the system by disincentivizing validator collusion and making corruption attempts economically unfeasible.**

**The bridge’s security is equal to that of the underlying blockchain it is bridging. Outside of consensus-level attacks on the underlying blockchain, user funds cannot be lost or stolen.**

* **Extensibility: seamless extension of the underlying protocol to other domains.**

The selection of chains for both users and developers, as well as different levels of difficulty for integrating an additional destination chain.

* **Generalizability: the protocol’s capability of handling arbitrary cross-domain data.**

**The capability of transferring information across multiple blockchains. This design enjoys strong network effects because of O(1) complexity — a single integration for a project gives it access to the entire ecosystem within the bridge. The drawback is that some designs usually trade off security and decentralization to get this scaling effect, which could have complex unintended consequences for the ecosystem.**

State-of-the-art interoperability protocols are not able to offer all the above properties but are limited to choosing two at the tradeoff of excluding another one.

For example, Light clients & Relays bridges (for example: IBC, Near Rainbow) are strong with **Generalizability** because header relay systems could pass around any kind of data. They are also strong with **Truestlessness** because they do not require additional trust assumptions, although there is a liveness assumption because a relayer is still required to transmit the information. And do not need any capital lockup. These strengths come at the cost of **Extensibility**. For each chain pair, developers must deploy a new light client smart contract on both the source and destination chain, which is somewhere between O(LogN) and O(N) complexity depends on the underline chains. This protocol also introduces latency and speed drawbacks caused by synchronizing data and validating proofs.

1. ***[Bonus]*** *Are there any projects that focus on solving trilemma similarly like Ethereum solves the scalability issue? If yes describe how it solves the problem.*

**[ANSWER]** Context and IBC (general bridge) are projects trying to solve this trilemma.

Like Ethereum that adds scalability via L2/sharding as a layer on top of an existing secure and decentralized backbone, Connext establishes the two most important properties first that are essential to the longevity of the protocol. In the case of interoperability, the two most important properties are maximum Trustlessness and Extensibility. The Connext NXTP offers these two properties and is specifically designed to be usable on any chain while still being as secure as the underlying domains. Then Generalizability is added by plugging in natively verified protocols on top of NXTP. This adds a “Layer 2” to Connext’s interoperability network.

### ****Ultra-light Clients****

*This week we focused on how to achieve ZK interoperability between chains. How we can use Zero Knowledge Proofs to sync a light client faster. This is especially useful for mobile first blockchains like Celo and is very useful for any blockchain.*

1. *Describe a specification for a light client based on zero-knowledge proofs. You should explain at least how to get the client synced up to the current state of the blockchain. Preferably go as far as explain how transaction inclusion proofs are generated too.*

**[ANSWER]**

Task: Light client that using ZKP to sync data and verify transaction inclusion.

Protocol

* Initialization:
  + When the light client first connects to a full-node server of the chain, the server needs to send ZK proofs and chain status data (merkle roots of recipient, state, and transactions) to the client to validate the current state, this can be done in one of 2 ways:
    1. For every block, a ZK-Snark proof is generated to proof:
       - Block is well-formed.
       - Each transaction in this block is valid by checking the transaction signature
       - For POW block, the nonce is correct per difficulty.
    2. Use a recursive ZK-Snark proof (like Mina) for all blocks up to date, this proof proves:
       - Like 1 above, the current block is valid
       - The ZK-Snark proof for previous blocks is valid
  + The light client verifies these proofs, once passed, stores chain status locally. For both approaches above, the status data is merkle roots of historical merkle roots of different data (transaction, state, recipient). For example: use Merkle Mountain Range data structure to stores transaction merkle roots (in each block) and only save transaction MMR root in the light client.

Note: since this is a non-interactive proof, any server can generate such proof and reused by other servers/clients after verification. This saves tremendous proof generate work.

* Synchronization:

We should not assume consistent connection between light client and network. After initialization, the later synchronization can be done periodically. Once connected, the client can ask server to only provide the proof of a specific block range. I.e., from say block 12345 to the latest. Similarly, this can be done in 2 ways,

* + 1. For every block (from 12345 to the latest), a ZK-Snark proof is generated to proof.
    2. Use a recursive ZK-Snark proof for all blocks up to date or a recursive ZK-Snark proof for a blocks range (from 12345 to the latest).

The light client then verifies and can update the local status (merkel roots of merkle roots) accordingly.

* Transaction Inclusion Verification:

To verify a transaction is included in a specific block by light client, a proof needs to be generated with following checks (public input: transaction data, transaction MMR root),

* + The transaction hash (computed from transaction data) is included in the transaction merkle root of a specific block.
  + This specific block is included in transaction MMR root.

1. *What is the relevance of light clients for bridge applications? How does it affect relayers?*

**[ANSWER]**

As described in question 1. 1, Light clients & Relays bridge is trustless bridge deploys light clients to facilitate cross-chain transactions. For example, in the token transfer use case, when a user locks their token on Ethereum, for minting the same amount of token on the harmony side, it requires Ethereum light client deployed on harmony to validate the lock transaction that happened on Ethereum. Similarly, when the user burns the minted token on harmony and tries to unlock the original tokens back to Ethereum, it requires the harmony light client deployed on Ethereum to validate the burn transaction that happened on harmony.

The relayer push status of one chain to another and vice visa through smart contract. And the status of a chain come from specific light client connect to it.

1. *Suppose code from* [*Plumo*](https://github.com/celo-org/plumo-prover) *is updated and was working in production on* [*Celo*](https://github.com/celo-org)*. What would be the main difficulty in porting Plumo over to Harmony?*

### ****Horizon Bridge****

*Horizon is Harmony’s bridge which allows crossing assets from Harmony to Ethereum/Binance and vice versa.*

1. *Check out* [*Horizon repository*](https://github.com/harmony-one/horizon)*. Briefly explain how the bridge process works (mention all necessary steps).*

*a) Comment the code for:*

* + [*harmony light client*](https://github.com/harmony-one/horizon/blob/main/contracts/HarmonyLightClient.sol)
  + [*ethereum light client*](https://github.com/harmony-one/horizon/blob/main/contracts/EthereumLightClient.sol)
  + [*token locker on harmony*](https://github.com/harmony-one/horizon/blob/main/contracts/TokenLockerOnHarmony.sol)
  + [*token locker on ethereum*](https://github.com/harmony-one/horizon/blob/main/contracts/TokenLockerOnEthereum.sol)
  + [*test contract*](https://github.com/harmony-one/horizon/blob/main/test/bridge.hmy.js)

*Provide commented code in your submission.*

**[ANSWER]**

*b) Why HarmonyLightClient has* ***bytes32 mmrRoot*** *field and EthereumLightClient does not? (You will need to think of blockchain architecture to answer this)*

**[ANSWER]**

1. ***[Bonus]*** *What are checkpoint blocks? Do they differ from epoch blocks and how? Why are they used?*

**[ANSWER]**

1. **[Infrastructure only]** *Horizon still doesn’t use zk-proofs in order to speed up light clients. What changes would you need to make to the code in order to apply initial state sync through zk-snarks? Provide pseudo code of improved version of light client.*

**[ANSWER]**

### ****Rainbow Bridge****

1. ***[Infrastructure only]*** *Comment code implemented in* [*NearBridge.sol*](https://github.com/aurora-is-near/rainbow-bridge/blob/master/contracts/eth/nearbridge/contracts/NearBridge.sol)*. Note that they're using fraud proofs and not zk proofs.*

**[ANSWER]**

1. *Explain the differences between Rainbow bridge and Horizon bridge. Which approach would you take when building your own bridge (describe technology stack you would use)?*

**[ANSWER]**

1. ***[Bonus]*** *Explain how merkle mountain ranges work and how they can be used in order to do block inclusion proofs. (You can check FlyClient for a light client implementation that uses MMR).*

**[ANSWER]** In addition to being a Merkle tree, an MMR allows efficient appends at the prover side and e\_ efficient block inclusion verifications at the verifier side. Further, it enables efficient subtree proofs, a proof that two MMRs agree on the first k leaves.

The full node can prove that a transaction was included in the longest chain by just providing an MMR proof (to prove that a block belongs to the longest chain) in addition to the current transaction proof (which shows that the transaction is included in the block).

### ****Thinking In ZK****

1. *If you have a chance to meet with the people who built the above protocols what questions would you ask them?*

**[ANSWER]**